EIE
Escuela de
Ingeniería Eléctrica

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE
COSTA RICA

IE-0117
Programación Bajo Plataformas Abiertas

Laboratorio #6

Fernando Jiménez Ureña B74020

II ciclo
Noviembre 2021

1. Resumen

En el presente laboratorio se estudian de manera teórica y práctica importantes conceptos que son esenciales para la programación como lo son los Compiladores, Depuración y la Automatización de proyectos. Además se dedica una sección a los punteros en C que conforman una parte indispensable en el aprendizaje y desarrollo en dicho lenguaje.

2. Nota Teórica

Es importante conocer varios conceptos para el mejor entendimiento y desarrollo de este laboratorio. Uno de los conceptos que más se desarrollará a lo largo del laboratorio son los compiladores. Un compilador es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.) en lenguaje de máquina. [1]

El comando *make* en Linux es una gran herramienta para facilitar la compilación de los programas que se van creando. Es un comando muy inteligente ya que logra reconocer qué archivos deben recompilarse, guarda los comandos de compilación con todos sus parámetros para encontrar librerías, ficheros de cabecera (.h), entre otros. Este comando se vuelve de gran utilidad cuando existen gran cantidad de ficheros fuente repartidos entre diferentes directorios. [2]

Luego, otro concepto importante es la depuración de programas con *gdb*. El proceso de depuración es aquel que se sigue para quitar errores en el código. Un depurador, como es el caso de *gdb*, es una herramienta de desarrollo muy especializada que se asocia a la aplicación en ejecución y permite inspeccionar el código. [3]

Finalmente, para este laboratorio se implementará el uso de punteros. Un puntero es una variable que contiene una dirección de memoria los cuales se definen mediante como por ejemplo *tipo * nombre* : donde nombre es el identificador de la variable puntero, y tipo se refiere al tipo de variable a la que apunta [4].

3. Análisis de Resultados

3.1. Compiladores

1. En su forma más general, en una sola palabra, ¿qué es un compilador?

   Traductor

2. Por medio de un diagrama, represente la principal diferencia entre un interprete y un compilador.

   En la Figura 1. se observa el diagrama mostrando las principales diferencias entre un intérprete y un compilador. La información contenida en dicho diagrama fue obtenida de [5].

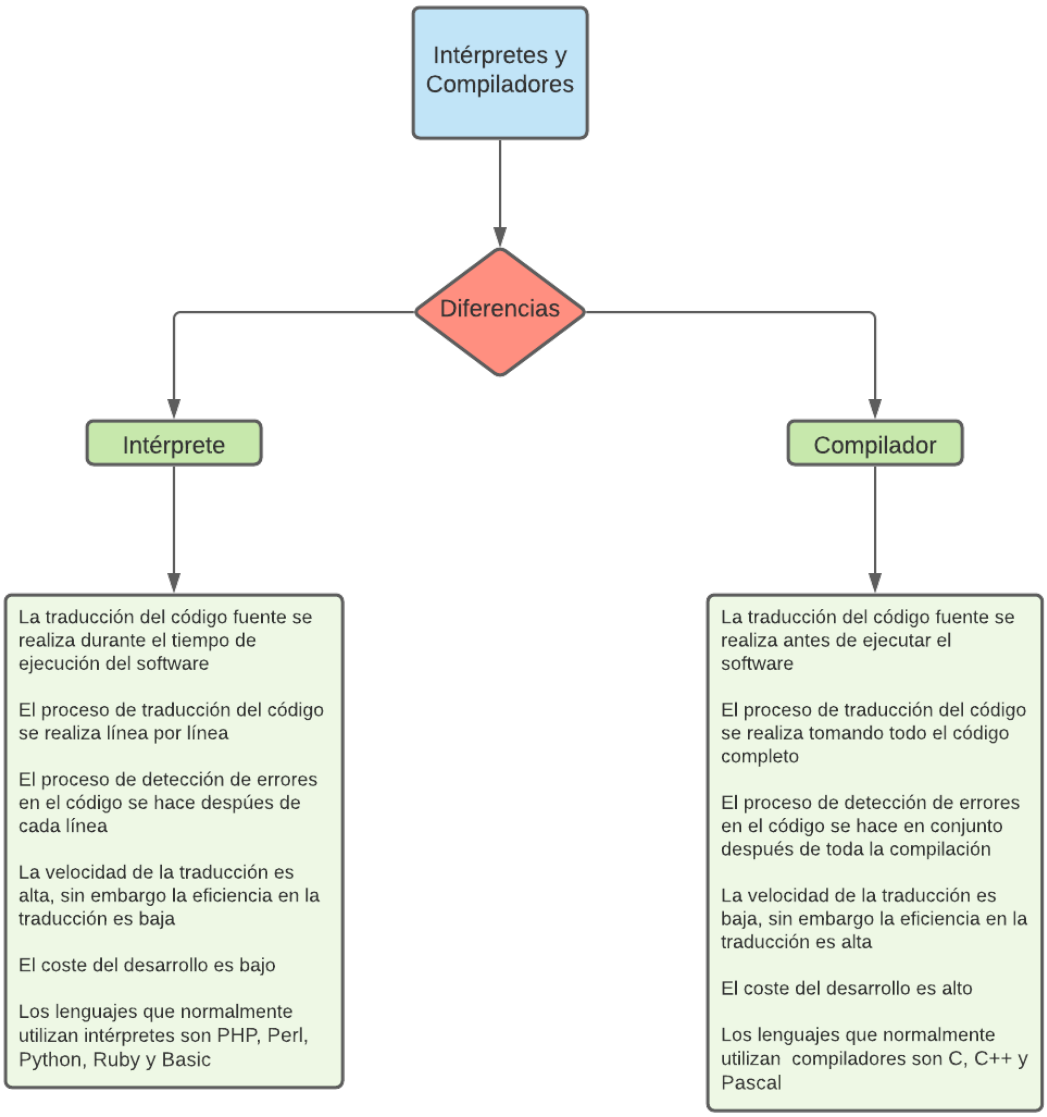

Figura 1: Diagrama sobre las diferencias entre un intérprete y un compilador

3. Explique brevemente con una o más razones por qué un lenguaje compilado puede ser más eficiente que uno interpretado.

   Se puede decir que es más eficiente el uso de un compilador por encima de un intérprete ya que una vez que se compila el código fuente, el compilador proporciona al procesador, el código máquina completo y ya listo para ser ejecutado. Esto hace que sea un proceso más robusto y una vez finalizada la compilación, se tenga toda la información a mano de cualquier error encontrado en el programa. A diferencia de un intérprete que hace el proceso de la traducción del código línea por línea, por lo que se detiene cada vez que encuentre un error en una línea y así sucesivamente por lo que puede volverse tedioso encontrar todos los errores finales en el programa, por lo que se puede decir que mediante un intérprete el proceso de traducción es poco eficiente y la velocidad en la ejecución es lenta [5]

4. ¿A qué se le conoce como cross-compiler?

   Un cross-compiler o compilador cruzado como se le conoce en español, es una herramienta bastante útil y poderosa en el desarrollo de software. Es un programa que es capaz de producir código ejecutable que se puede ejecutar en una plataforma que actualmente no es la plataforma residente para el compilador. Esto es bastante útil cuando el desarrollador necesita distribuir su programa en distintas plataformas ya que le permite mejorar el manejo de las funciones informáticas y le ayuda a superar barreras como cuando ciertos computadores en un sistema integrado posee una cantidad limitada de recursos ya que logra crear una ejecución interrelacionada entre varios componentes del sistema. [6]

5. Enumere los 5 elementos que componen la estructura típica de un compilador.

   a) Análisis Léxico

   b) Parser / Análisis sintáctico

   c) Análisis semántico

   d) Optimizaciones

   e) Generación de código

6. Describa brevemente la función que realiza, en el proceso de compilación, cada uno de los elementos mencionados en el punto anterior.

   a) Análisis Léxico: Reconoce las palabras provenientes del código fuente y las separa en tokens.

   b) Parser / Análisis sintáctico: En esta sección de la compilación, el compilador entiende la estructura de las oraciones por lo que logra reconocer la función de cada palabra. Finalmente se procede a agrupar las palabras en estructuras de alto nivel.

   c) Análisis semántico: Es el proceso más complicado para el compilador ya que necesita interpretar de alguna manera lo que un usuario quiere decir en el código

fuente. Esta parte del proceso lleva un análisis semántico bastante limitado e intenta encontrar inconsistencias.

d) Optimizaciones: Este proceso es el que intenta hacer el programa lo más 'económico' posible, ya que hace que el programa corra más rápido y al mismo tiempo hace que utilice menos memoria, menos potencia requerida, menos accesos a base datos y limita el uso de la red

e) Generación de código: Esta es la parte final en el proceso de compilación ya que aquí es donde se genera el código objeto

3.2. Depuración: gdb

1. ¿Cuál es la bandera del compilador gcc que se utiliza para generar un código que tenga los componentes necesarios para depurar (debugging)?

   La bandera que se utiliza para generar un código que tenga los componentes necesarios para depurar en el compilador gcc es *-g*. Un depurador, en este caso *-g* se utiliza para detectar errores de ejecución en un programa en C que ha compilado sin problemas. Este depurador ejecuta el programa línea a línea mostrando al mismo tiempo el código fuente y el valor de las variables. [7]

2. ¿Cuál es la línea de comandos en terminal necesaria para compilar con gcc un código fuente con el nombre fuente.c en un ejecutable llamado bin capaz de ser depurado con gdb?

   gcc -g -o bin fuente.c

3. Describa brevemente para que sirven los siguientes comandos en gdb:

   a) run: Este comando inicia el programa corrido bajo gdb. Cabe destacar que el programa que se inicia es el que se ha seleccionado previamente con el comando de archivo, o en la línea de comando de Unix cuando se inició gdb. [8]

   b) break: El comando break se utiliza como un punto de detenimiento donde se desea que el programa se detenga temporalmente en la ejecución para poder verificar en que momento el programa está teniendo errores o para chequear los valores de las variables. [8]

   c) quit: Este comando es utilizado para salir de GDB

   d) continue: Este comando se utiliza para que el programa siga corriendo luego de haberlo detenido en un punto de detenimiento. [8]

   e) print: Este comando se encarga de imprimir el valor de la expresión que puede ser por ejemplo el nombre de una variable. Se pueden agregar argumentos luego del comando para imprimir únicamente un intervalo especifico de líneas por ejemplo las primeras 25 líneas del programa. [8]

Luego, se procede a a realizar la depuración del programa *serie.c* brindado en el enunciado del laboratorio. Primero como se puede apreciar en la Figura 2., se intentó compilar de

manera normal el programa con gcc y su argumento *-g* y se puede observar que se detectaron múltiples errores en el programa.

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ gcc -g seri
e.c -o serie
serie.c: In function 'main':
serie.c:31:9: error: expected ';' before string constant
   31 |   printf"(x^0)/0! + (x^1)/1! + (x^2)/2! + (x^3)/3! + (x^4)/4! + ........ + (x^n)/n! \n");
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      |         ;
serie.c:31:88: error: expected statement before ')' token
   31 |   printf"(x^0)/0! + (x^1)/1! + (x^2)/2! + (x^3)/3! + (x^4)/4! + ........ + (x^n)/n! \n");
      |                                                                                        ^
serie.c:35:11: warning: format '%f' expects argument of type 'float *', but argument 2 has type 'double *' [-Wformat=]
   35 |   scanf("%f",&x);
      |          ~^  ~~
      |           |  |
      |           |  double *
      |           float *
      |          %lf
serie.c:39:16: error: expected ')' before ';' token
   39 |   scanf("%d",&n;
      |               ^
      |               )
serie.c:44:12: error: expected ';' before '}' token
   44 |   return 0;
      |            ^
      |            ;
   45 | }
      | ~
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$
```

Figura 2: Compilación y errores hallados con el compilador gcc

Se deben realizar las correcciones mostradas en la terminal para que se pueda generar el archivo ejecutable. Este archivo ejecutable es vital debido a que para utilizar el depurador gbd, éste únicamente detecta archivos ejecutables para poder realizar el proceso de depuración.

Luego de realizar las debidas correcciones, se puede apreciar en la Figura 3. que se logró crear correctamente el archivo ejecutable.

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ gcc -g seri
e.c -o serie
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ls
README.md  serie  serie.c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$
```

Figura 3: Ejecutable *serie* creado correctamente

Se procede a ejecutar el programa, y se puede observar en la Figura 4. que efectivamente si se logra correr. Sin embargo una vez que se introducen los valores solicitados por el programa, la serie siempre dará como resultado *inf*. Por lo que en este caso, es ideal depurar el programa para encontrar qué está fallando a pesar de que el programa pueda ser ejecutado en la terminal.

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ./serie
Este programa calcula el valor de la serie:
x^0)/0! + (x^1)/1! + (x^2)/2! + (x^3)/3! + (x^4)/4! + ........ + (x^n)/n!
Ingrese el valor de x :
2

Introduzca un valor entero para n :
3
El resultado de la serie para los valores ingresados es inf
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ./serie
Este programa calcula el valor de la serie:
x^0)/0! + (x^1)/1! + (x^2)/2! + (x^3)/3! + (x^4)/4! + ........ + (x^n)/n!
Ingrese el valor de x :
5

Introduzca un valor entero para n :
25
El resultado de la serie para los valores ingresados es inf
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$
```

Figura 4: Ejecutable *serie* creado correctamente

Se procede a ejecutar el depurador GBD para analizar más a fondo cuál es la raíz de este problema. En la Figura 5. se selecciona un *breakpoint* en la línea 41 donde se encuentra la expresión 'double valorserie = CalcularValorSerie(x, n);'

```
(gdb) b 41
Breakpoint 1 at 0x80012d4: file serie.c, line 41.
(gdb)
```

Figura 5: Eligiendo el breakpoint

Luego, se procede a utilizar el comando *run* y se puede observar como el depurador ejecuta el programa y pide los valores al usuario. En este caso se utilizarán los valores de x = 2 y n = 3 que se espera que devuelva el valor de 5. Una vez ejecutado como se aprecia en la Figura 6., que el programa se detiene en el breakpoint que se eligió anteriormente.

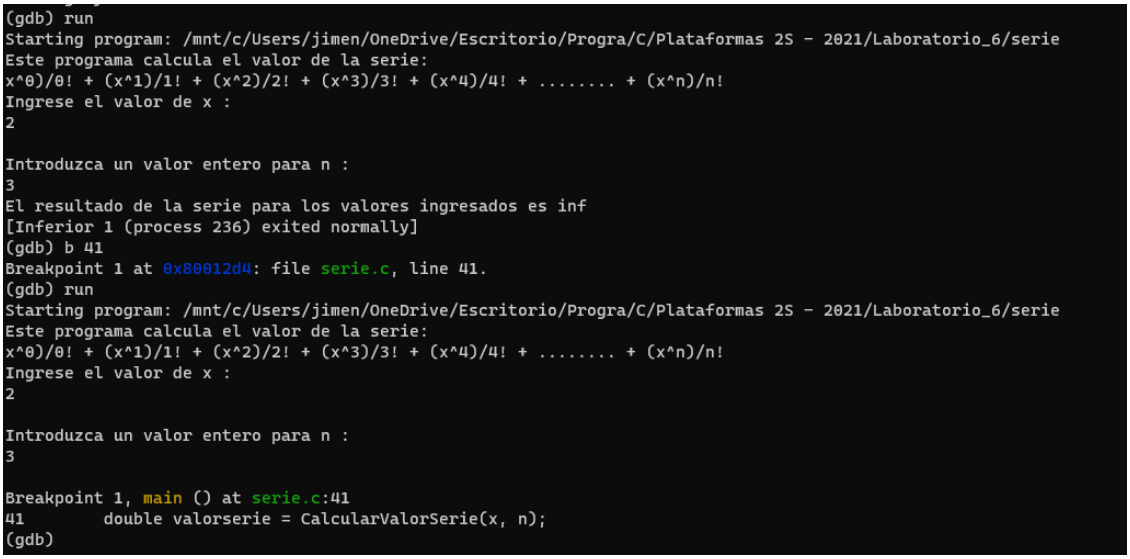

Figura 6: Ejecutando el depurador gdb

Se procede a entrar a la función CalcularValorSerie() mediante el comando *step* y utilizando también el comando *next* hasta llegar a la función CalcularFactorial(). En la Figura 7. se observa la manera ne que se llegó a esta última función y además se puede observar mediante el comando *backtrace* o *bt* en cual lugar nos encontramos en ese momento.

```
(gdb) step
CalcularValorSerie (x=6.9533553009897577e-310, n=0) at serie.c:16
16      double CalcularValorSerie(double x, int n) {
(gdb) next
17          double valor = 0.0;
(gdb) nezt
Undefined command: "nezt".  Try "help".
(gdb) next
18          double xpow = 1;
(gdb) next
20          for (int k = 0; k <= n; k++) {
(gdb) next
21              valor += xpow / CalcularFactorial(k);
(gdb) next
22              xpow = xpow * x;
(gdb) step
20          for (int k = 0; k <= n; k++) {
(gdb) next
21              valor += xpow / CalcularFactorial(k);
(gdb) s
CalcularFactorial (numero=0) at serie.c:6
6       {
(gdb) bt
#0  CalcularFactorial (numero=0) at serie.c:6
#1  0x000000000800121c in CalcularValorSerie (x=2, n=3) at serie.c:21
#2  0x00000000080012e7 in main () at serie.c:41
(gdb)
```

Figura 7: Entrando a la función y uso del comando backtrace

Finalmente se sigue recorriendo el programa con el comando *next* o abreviado *n* y se llega hasta al valor de la variable 'fact' donde utilizando el comando *print* se puede concluir que el valor de dicha variable nunca cambia y esto es lo que está generando el problema. Esto puede observarse en la Figura 8.

Observando más a fondo, en la operación 'fact = fact * j', 'fact' a la hora de ser inicializada en 0 esta operación nunca podrá avanzar y se concebirá el error. Por lo que 'fact' debe ser inicializada en 1 para que el ciclo for pueda utilizar los valores correctos a la hora de ser ejecutado.

```
#0  CalcularFactorial (numero=32767) at serie.c:6
#1  0x000000000800121c in CalcularValorSerie (x=2, n=3) at serie.c:21
#2  0x00000000080012e7 in main () at serie.c:41
(gdb) s
7         int fact = 0;
(gdb) s
9         for (int j = 1; j <= numero; j++) {
(gdb) print fact
$1 = 0
(gdb) n
13        return fact;
(gdb)
```

Figura 8: Analizando la variable *fact*

Ya una vez encontrado el error, se procede a corregirlo y en la línea 7 del código fuente se procede a cambiar el valor de 'fact' a igual a 1. Se procede a compilar nuevamente el programa y a realizar la misma prueba con los valores de x = 2 y n = 3.

```
1   #include <stdio.h>
2   #include <math.h>
3
4
5   int CalcularFactorial(int numero)
6   {
7     int fact = 1;
8
9     for (int j = 1; j <= numero; j++) {
10      fact = fact * j;
11    }
12
13    return fact;
14  }
```

Figura 9: Arreglando el valor de la variable *fact*

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ./serie
Este programa calcula el valor de la serie:
x^0)/0! + (x^1)/1! + (x^2)/2! + (x^3)/3! + (x^4)/4! + ........ + (x^n)/n!
Ingrese el valor de x :
2

Introduzca un valor entero para n :
3
El resultado de la serie para los valores ingresados es 6.333333
```

Figura 10: Compilación y ejecución del programa

Como se puede observar en la Figura 9. y Figura 10. luego de realizar el cambio, finalmente el programa logra retornar el valor deseado y se arregló el problema.

3.3. Punteros

3.3.1. Punteros Simples

Para este problema, se desarrolló todo en un mismo programa sin embargo dividiéndolo en 3 funciones diferentes: La función *CambioPorValor* donde se implementó un algoritmo acudiendo a una variable auxiliar encargada de hacer el cambio del valor de las variables, luego la función *CambioPorPuntero* realiza un proceso similiar sin embargo acudiendo al uso de punteros y finalmente la función *main* se encarga de llamar a las dos funciones y de pasarles los parámetros e imprimir el resultado del intercambio.

Se puede observar en las siguientes Figuras el código implementado y que efectivamente si se logró resolver el problema planteado.

```
//Prototipos de Funciones

void CambioPorValor(int,int);

void CambioPorPuntero(int *, int *);

int a = 10, b = 75, c = 45, d = 90;
```

Figura 11: Prototipo de las funciones y declaración de variables globales

```c
void CambioPorValor(int a, int b){

    int aux;

    aux = a;
    a = b;
    b = aux;

}

void CambioPorPuntero(int *dirNum1, int *dirNum2){

    int aux;

    //Intercambiar los valores de los variables

    aux = *dirNum1;
    *dirNum1 = *dirNum2;
    *dirNum2 = aux;
}
```

Figura 12: Funciones para desarrollar el problema

```c
int main(){

    printf("\nUtilizando el método por Valor\n\n");

    printf("El valor del primer numero es %d\n", a);
    printf("El valor del segundo numero es %d\n", b);

    CambioPorValor(a,b);

    printf("El nuevo valor del primer numero es %d\n", b);
    printf("El nuevo valor del segundo numero es %d\n", a);

    printf("\n-------------------------\n");

    printf("\nUtilizando el método de Punteros\n\n");

    printf("El valor del primer numero es %d\n", c);
    printf("El valor del segundo numero es %d\n", d);

    CambioPorPuntero(&c,&d);

    printf("El nuevo valor del primer numero es %d\n", c);
    printf("El nuevo valor del segundo numero es %d\n", d);

    return 0;
}
```

Figura 13: Función *main*

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ gcc -o exe
puntero_simple.c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ./exe

Utilizando el método por Valor

El valor del primer numero es 10
El valor del segundo numero es 75
El nuevo valor del primer numero es 75
El nuevo valor del segundo numero es 10

-------------------------

Utilizando el método de Punteros

El valor del primer numero es 45
El valor del segundo numero es 90
El nuevo valor del primer numero es 90
El nuevo valor del segundo numero es 45
```

Figura 14: Resultados Obtenidos

3.3.2. Punteros, funciones y matrices

Para este problema, se implementaron dos funciones: *armarMatriz* y *mostrarMatriz*. En estas funciones se genera la matriz 10x10 con los numeros aleatorios y luego se imprime en pantalla como se aprecia en la Figura 15. y Figura 16. respectivamente

```c
void armarMatriz(){

//En esta funcion se definen las variables para generar los numeros aleatorios

    int aleatorio;

//Primero se desea generar el numero aleatorio.

    srand(time(NULL)); //genera el numero


//Este ciclo for se utiliza para rellenar la matriz
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            //la formula para generar el numero aleatorio es:
            // variable = limite_inferior + rand()% (limite_superior +1 - limite inferior)
            //En este caso se utiliza un rango de 1 a 100
            aleatorio = 1 + rand()%(100);
            matriz[i][j] = aleatorio; // = *(*(puntero_matriz+i)+j); //puntero_matriz[i][j];
        }
    }
}


void mostrarMatriz(){

    printf("\n\nImprimiendo Matriz:\n");
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            printf(" %d", matriz[i][j]); printf(" ");
        }
        printf("\n");
    }
}
```

Figura 15: Funciones para generar la matriz de números aleatorios

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ gcc -o exe
punteros_funciones_matrices.c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Laboratorio_6$ ./exe


Imprimiendo Matriz:
 29  32  55  93  70  29  24  93  59  45
 66  95  59  35  51  57  79  23  28  100
 5  28  55  60  75  95  14  22  49  27
 15  29  10  21  22  79  1  97  23  12
 41  88  6  51  74  8  7  53  31  86
 4  87  65  58  98  40  52  11  61  52
 89  27  81  98  47  54  28  47  50  50
 58  42  89  15  92  63  23  99  67  5
 84  70  91  1  79  88  40  82  50  100
 33  38  26  65  87  24  18  14  70  67
```

Figura 16: Matriz Impresa

Es importante notar que no se logró resolver en su totalidad el problema ya que no se pudo encontrar la manera de definir la función *buscar*. Por lo que únicamente se obtuvo las funciones para generar la raíz de números aleatorios e imprimirla.

4. Conclusiones y Recomendaciones

Para concluir es importante reconocer la importancia de haber puesto en práctica conceptos como la depuración de un programa, la automatización de proyectos con el uso del comando *make*, así como reconocer las diferencias entre un compilador y un intérprete. Esto logra desarrollar buenos hábitos de programación así como la facilitación del mismo y ahorrar tiempo y recursos. Además, se lograron desarrollar diversos ejercicios con Punteros que fueron de gran utilidad para el entendimiento de este tema.

5. Referencias

[1] Calvo, J. (2021). ¿Que es un Compilador en programación? – Blog Europeanvalley. Europeanvalley.es. Retrieved 16 November 2021, from https://www.europeanvalley.es/noticias/que-es-un-compilador-en-programacion/.

[2] Conceptos básicos de make y los Makefile. Chuidiang.org. (2021). Retrieved 17 November 2021, from http://www.chuidiang.org/clinux/herramientas/makefile.phpresumen.

[3] ¿Qué es la depuración? - Visual Studio (Windows). Docs.microsoft.com. (2021). Retrieved 17 November 2021, from https://docs.microsoft.com/es-es/visualstudio/debugger/what-is-debugging?view=vs-2022.

[4] A.Serrano.Ïntroduccion al lenguaje C". Recuperado de https://blog.utp.edu.co/ jnsanchez/files/2011/03 al-lenguaje-c.pdf

[5] Compilador e intérprete: definición y diferencias. IONOS Digitalguide. (2021). Retrieved 15 November 2021, from https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/compilador-e-interprete/.

[6] ¿Qué es un compilador cruzado?. Netinbag.com. (2021). Retrieved 15 November 2021, from https://www.netinbag.com/es/internet/what-is-a-cross-compiler.html.

[7] La opción de depuración. It.uc3m.es. (2021). Retrieved 15 November 2021, from http://www.it.uc3m.es/p

[8] GDB Tutorial. Web.eecs.umich.edu. (2021). Retrieved 16 November 2021, from https://web.eecs.umich.e gih/pointers/summary.html.